


Unisenac
Campus Pelotas



Algoritmos e Estruturas de Dados I


CENTRO UNIVERSITÁRIO UNISENAC – CAMPUS PELOTAS
CURSOS SUPERIORES: ESCOLA DE TECNOLOGIA
PROF. EDÉCIO FERNANDO IEPSSEN

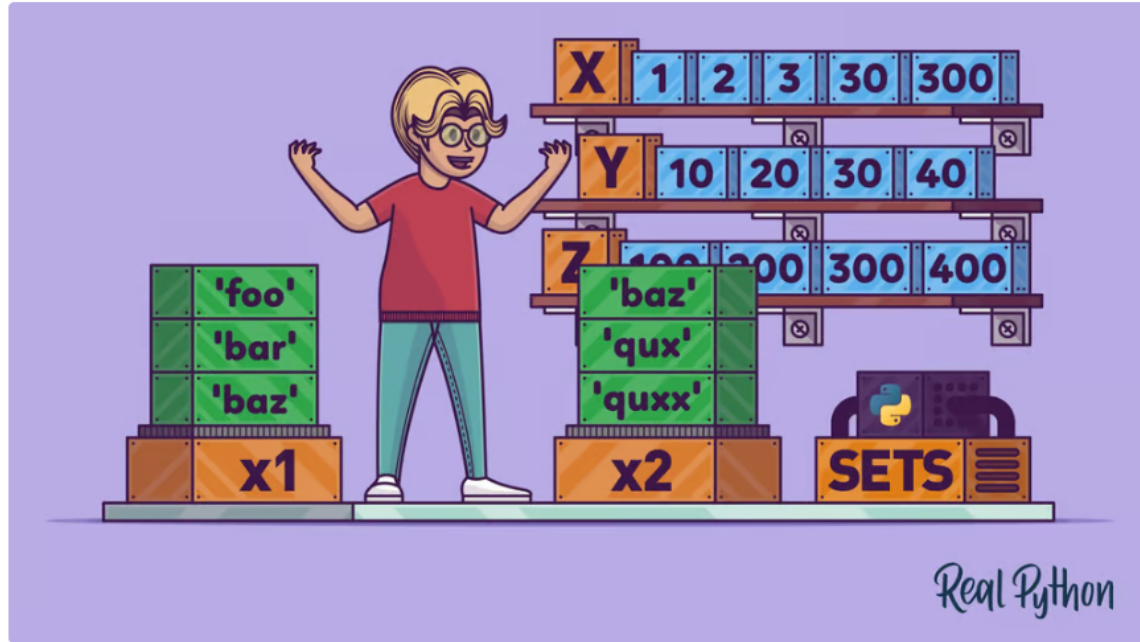
Conjuntos

Os conjuntos em Python, também conhecidos como sets, **são coleções de elementos únicos que não são ordenados**. São uma estrutura de dados fundamental da linguagem e são úteis para garantir que não haja itens duplicados em uma coleção. 

Conjuntos em Python

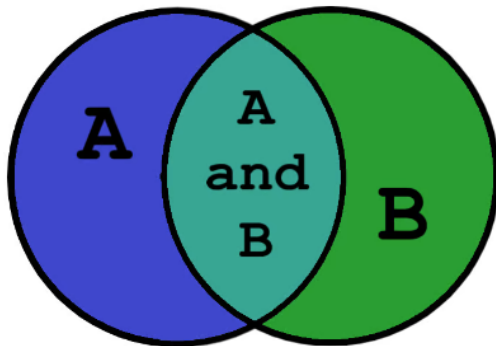
Definição	Coleção não ordenada de elementos únicos
Como definir	Chaves {} e elementos separados por vírgulas
Operações	União, interseção e diferença
Métodos	remove, discard, pop
Limitações	Não suportam indexação ou fatiamento direto

Para criar um conjunto vazio, é necessário usar a função `set()`, pois `{}` cria um dicionário vazio. 



Sets in Python

Talvez você se lembre de ter aprendido sobre **conjuntos** e **teoria de conjuntos** em algum momento da sua educação matemática. Talvez você até se lembre dos diagramas de Venn:



Se isso não lhe parece familiar, não se preocupe! Este tutorial ainda deve ser facilmente acessível para você.

Em matemática, uma definição rigorosa de um conjunto pode ser abstrata e difícil de entender. Na prática, porém, um conjunto pode ser pensado simplesmente como uma coleção bem definida de objetos distintos, normalmente chamados de **elementos** ou **membros**.

Agrupar objetos em um conjunto pode ser útil também na programação, e o Python fornece um tipo de conjunto integrado para fazer isso. Conjuntos são diferenciados de outros tipos de objetos pelas operações únicas que podem ser executadas neles.

Definindo um conjunto

O tipo interno do Python settem as seguintes características:

- Os conjuntos não são ordenados.
- Elementos definidos são únicos. Elementos duplicados não são permitidos.
- Um conjunto em si pode ser modificado, mas os elementos contidos no conjunto devem ser de um tipo **imutável**.

Vamos ver o que tudo isso significa e como você pode trabalhar com conjuntos em Python.

Um conjunto pode ser criado de duas maneiras. Primeiro, você pode definir um conjunto com a função interna `set()`:

Pitão

```
x = set(<iter>)
```

Neste caso, o argumento `<iter>` é um iterável — novamente, por enquanto, think list ou tuple — que gera a lista de objetos a serem incluídos no conjunto. Isso é análogo ao `<iter>` argumento dado ao `.extend()` método list:

Pitão



```
>>> x = set(['foo', 'bar', 'baz', 'foo', 'qux'])
>>> x
{'qux', 'foo', 'bar', 'baz'}
```

`Strings` também são iteráveis, então uma string pode ser passada para `set()` também. Você já viu que `list(s)` gera uma lista dos caracteres na string `s`. Similarmente, `set(s)` gera um conjunto dos caracteres em `s`:

Pitão

```
>>> s = 'quux'

>>> list(s)
['q', 'u', 'u', 'x']
>>> set(s)
{'x', 'u', 'q'}
```

Você pode ver que os conjuntos resultantes são desordenados: a ordem original, conforme especificado na definição, não é necessariamente preservada. Além disso, valores duplicados são representados no conjunto apenas uma vez, como com a string `'foo'` nos dois primeiros exemplos e a letra `'u'` no terceiro.

Alternativamente, um conjunto pode ser definido com chaves (`{}`):

Pitão

```
x = {<obj>, <obj>, ..., <obj>}
```

Quando um conjunto é definido dessa forma, cada um `<obj>` se torna um elemento distinto do conjunto, mesmo que seja um iterável. Esse comportamento é semelhante ao do `.append()` método `list`.

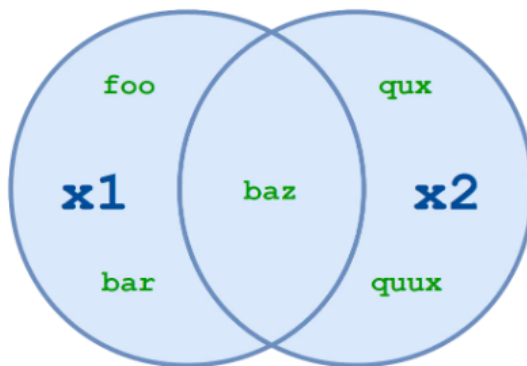
Operadores e métodos disponíveis

Abaixo está uma lista das operações de conjunto disponíveis em Python. Algumas são realizadas por operador, algumas por método e algumas por ambos. O princípio descrito acima geralmente se aplica: onde um conjunto é esperado, os métodos normalmente aceitarão qualquer iterável como argumento, mas os operadores exigem conjuntos reais como operandos.

```
x1.union(x2[, x3 ...])
```

```
x1 | x2 [| x3 ...]
```

Calcule a união de dois ou mais conjuntos.



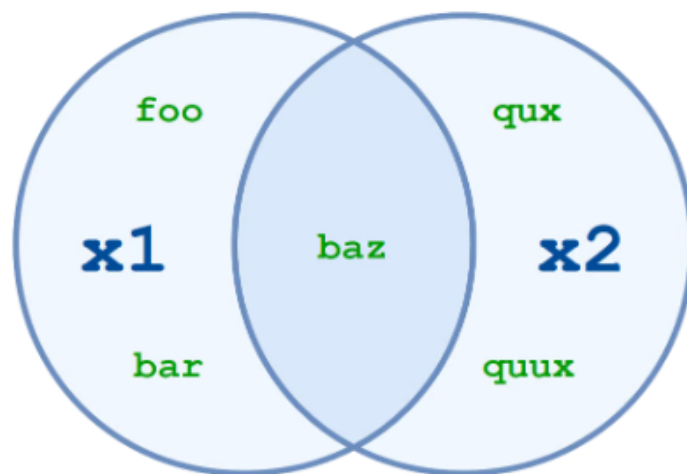
Definir União

`x1.union(x2)` e `x1 | x2` ambos retornam o conjunto de todos os elementos em `x1` ou `x2`:

```
x1.intersection(x2[, x3 ...])
```

```
x1 & x2 [& x3 ...]
```

Calcule a interseção de dois ou mais conjuntos.



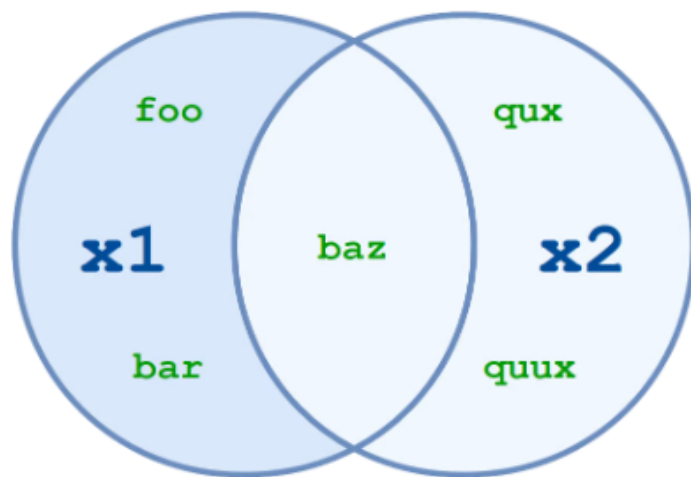
Definir Intersecção

`x1.intersection(x2)` e `x1 & x2` retornar o conjunto de elementos comuns a ambos `x1` e `x2`:


```
x1.difference(x2[, x3 ...])
```

```
x1 - x2 [- x3 ...]
```

Calcule a diferença entre dois ou mais conjuntos.



Definir diferença

`x1.difference(x2)` e `x1 - x2` retornar o conjunto de todos os elementos que estão em `x1` mas não estão em `x2`:

Exemplos e Exercícios:

A partir do DataSet de pilotos vitoriosos na Fórmula 1:

- Ler nome de um piloto. Exibir em quais anos (sem repetições) este piloto venceu corridas.
- Ler um ano. Exibir quais pilotos (sem repetições) venceram corridas. Indicar se algum destes pilotos teve a sua primeira vitória neste ano.

Exemplos e Exercícios:



A partir do DataSet de Visitantes Estrangeiros no Japão:

- Exibir o Top 10 países com maior número de visitantes.
- Comparar 2 países. Ler 2 países e mostrar o total de visitantes de cada um deles.
- Analisar 2 anos. Ler 2 anos e mostrar o top 10 de cada ano. Exibir também se há algum país diferente no top 10 em cada ano.